9 Abstract

A system is described which allows buyers to define their preferences and sellers to define their capabilities, then determines which trading points maximize the utility of the buyer. The system suggests trades by exploiting the flexibilities and tradeoffs encoded by both parties, thus providing win-win trades. A second level of optimization ranks the trades with all suppliers, allowing the buyer to rapidly determine the best alternatives. The system allows for rich negotiation spaces and supports continuous, discrete, and range or interval decision factors.

Copyright © 1996, by the author(s).

A Distance Function for Range Variables

The distance between buyer range variable $r_i = (\underline{r}_i, \overline{r}_i)$ and seller range variable $r_j = (\underline{r}_j, \overline{r}_j)$ is related to the overlap defined as

$$\text{overlap}(r_i, r_j) = \int_{\mu_j - \sigma_j}^{\mu_j + \sigma_j} dx\, \mathcal{N}(x; r_i)\mathcal{N}(x; r_j)$$

where $\mathcal{N}(x; r_i)$ is a Gaussian in x centered at $\mu_i = (\underline{r}_i + \overline{r}_i)/2$ with standard deviation $\sigma_i = \alpha(\overline{r}_i - \underline{r}_i)$ and α is a tunable parameter. The range of integration is bounded by the seller's capabilities. Evaluating the integral, we find

$$\text{overlap}(r_i, r_j) = \frac{1}{2\sqrt{2\pi(\sigma_i^2 + \sigma_j^2)}} \exp\left[-\frac{(\mu_i - \mu_j)^2}{2(\sigma_i^2 + \sigma_j^2)}\right] \left(\text{erf}\left[\frac{\sigma_j^2 + \sigma_j\mu_i - \sigma_j\mu_j + \sigma_i^2}{\sigma_i\sqrt{2(\sigma_i^2 + \sigma_j^2)}}\right] - \text{erf}\left[\frac{-\sigma_j^2 + \sigma_j\mu_i - \sigma_j\mu_j - \sigma_i^2}{\sigma_i\sqrt{2(\sigma_i^2 + \sigma_j^2)}}\right]\right).$$

By adjusting α we can control the hardness of the boundary at the ends of the range (smaller α corresponds to sharper dropoff outside the range). The maximum overlap will occur when $\mu_j = \mu_i$ and $\sigma_j = 0$. Substituting those values in the above equation, we get

$$\text{maxOverlap} = \frac{\text{erf}\left[\frac{1}{\sqrt{2}}\right]}{\sqrt{2\pi(\sigma_i^2 + \sigma_j^2)}}$$

The distance $d(r_i, r_j)$ is related to the overlap so that it varies from 0 (for identical range variables) to ∞ (for maximally distant range variables). Consequently, we define

$$R(r_i, r_j) = -\ln\left[\frac{\text{overlap}(r_i, r_j)}{\text{maxOverlap}}\right].$$

B From Rankings to Numerical Values

The numerical values in the tables $Z(\varkappa_i, \varkappa_j)$ can be formed from the buyer's rankings in a number of ways. Since we are minimizing distance, we want preferred options to contribute little Z and progressively less favorable options to have progressively larger Z values. To approach this, we first consider the problem of defining a Z' value normalized to lie between 0 and 1 and then take $Z = -\ln[1 - Z']$.

One possibility for Z' is a linear scaling. If d_i is the integer indicating the number of values in $\mathcal{D}_i$, $r_i^{(\alpha)}(\varkappa_j)$ indicates the buyer-specified rank (smaller rank means more preferable) for the option $\varkappa_i = \alpha \in \mathcal{D}_i$ conditional on $\varkappa_j$, then we might set $Z'(\varkappa_i = \alpha, \varkappa_j) = (r_i^{(\alpha)}(v_j) - 1)/(d_i - 1)$. Alternatively, we might use a non-linear scaling and set $Z(\varkappa_i = \alpha, v_j) = (1 - \beta^{r_i^{(\alpha)}(\varkappa_j)-1})/(1 - \beta^{d_i - 1})$ for some $\beta \in [0, 1]$ (exponential) or set $Z(v_i = \alpha, v_j) = \left((r_i^{(\alpha)}(v_j) - 1)/(R_i - 1)\right)^n$ for some $n > 0$ (algebraic). The corresponding scaling functions for Z are given in Table 4.

type	Z′
linear	$\ln\left[\dfrac{R_i-1}{R_i-r_i^{(\alpha)}(v_j)}\right]$
exponential	$\ln\left[\dfrac{1-\beta^{R_i-1}}{\beta^{r_i^{(\alpha)}(v_j)-1}-\beta^{R_i-1}}\right]$
algebraic	$-\ln\left[1-\left(\dfrac{r_i^{(\alpha)}(v_j)-1}{R_i-1}\right)^n\right]$

Table 4: Possible numerical ranking functions.

For those allowed values of $\varkappa_i$ which are left unranked by the buyer, we can assign them a common numerical value greater than the lowest ranked value. For example if $r_i(\varkappa_j)$ is the largest ranked value we might assign all remaining $\varkappa_i$ values to have $Z(\varkappa_i,\varkappa_j) = r_i(\varkappa_j)/(d_i-1)$ in the linear case or $Z(\varkappa_i,\varkappa_j) = (1-\beta^{r_i(\varkappa_j)})/(1-\beta^{d_i-1})$ or $Z(\varkappa_i,\varkappa_j) = (1-1/(r_i(\varkappa_j)+1))/(1-1/d_i)$ in the non-linear cases.

C Determination of Scaling Factors

In this section we supply the details necessary to determine the scaling factors Q_r and Q_d. In the equations defining the average distances we have need of the integrals $\int_V d\mathbf{u}\ \exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})]$ and $\int_V d\mathbf{u}\ (\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})\exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})]$. In general, these integrations are not analytically tractable; however we will assume that the region of integration defined by $\overline{x}$ and $\underline{x}$ extends at least 2 standard deviations out in all directions so that the volume of integration is effectively infinite. In this case we can evaluate the integrals in closed form:

$$\int_V d\mathbf{u}\ \exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})] \approx \sqrt{\frac{\pi^{n_c}}{\det\mathbf{C}^{-1}}}$$

and

$$\int_V d\mathbf{u}\ (\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})\exp[-(\mathbf{u}-\tilde{\mathbf{u}})^t\mathbf{C}^{-1}(\mathbf{u}-\tilde{\mathbf{u}})] \approx \frac{n_c}{2}\sqrt{\frac{\pi^{n_c}}{\det\mathbf{C}^{-1}}}.$$

To this point we have not been specific about what is meant by the average over the range variables. In fact, we will assume that, however we define this average, it is independent of what the buyer requested and thus can be removed from inside the sum over the discrete variables. In this case we will concentrate on setting the discrete and continuous components equal. We have $\langle d_c\rangle/n_c = \langle d_d\rangle/n_d$ where

$$\frac{\langle d_d\rangle}{n_d} \equiv \frac{Q_d}{n_d}\frac{\sum_{\mathbf{v}} d_d\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]} \quad \text{and} \quad \frac{\langle d_c\rangle}{n_c} \equiv \frac{1}{n_c}\frac{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ d_c\exp[-d_c]}{\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c]}$$

and cancelling denominators Q_d is determined by solving

$$\frac{Q_d}{n_d}\sum_{\mathbf{v}} d_d\exp[-Q_d d_d]\int_V d\mathbf{u}\ \exp[-d_c] = \frac{1}{n_c}\sum_{\mathbf{v}}\exp[-Q_d d_d]\int_V d\mathbf{u}\ d_c\exp[-d_c].$$

This can be simplified further given our assumptions about the Gaussian integral:

$$\frac{Q_d}{n_d}\sum_{\mathbf{v}}\frac{d_d\exp[-Q_d d_d]}{\sqrt{\det\mathbf{C}^{-1}(\mathbf{v})}}=\frac{1}{2}\sum_{\mathbf{v}}\frac{\exp[-Q_d d_d]}{\sqrt{\det\mathbf{C}^{-1}(\mathbf{v})}}.$$

In most cases there will also be at least one finite root, which is what we need. If there are a multiplicity of roots, we select that root having the smallest value to ensure that the distribution defined by the utility function is as flat as possible. If there are no real roots, then we select the Q_d which minimizes the error.

D Phase 1 Optimization

In this section we detail the solution to the phase 1 optimization problem defined in Eq. (5).

D.1 General Formulation

We begin by writing down the objective function:

$$d_1(\mathbf{x},\boldsymbol{\varkappa})=\left(\mathbf{x}^{(s)}-\boldsymbol{\mu}(\boldsymbol{\varkappa})\right)^t\mathbf{C}_w^{-1}\left(\mathbf{x}^{(s)}-\boldsymbol{\mu}(\boldsymbol{\varkappa})\right)+R(\mathbf{r};\boldsymbol{\varkappa})$$

where (see Eq. (8))

$$\mathbf{x}^{(s)}=\left(\mathbf{I}-\mathbf{M}^{(s)}(\boldsymbol{\varkappa}^{(s)},\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\boldsymbol{\varkappa}^{(s)},\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\})+\left(\mathbf{I}-\mathbf{M}^{(s)}(\mathbf{v}^{(s)},\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{u}^{(b)} \tag{8}$$

as long as $x_k^{(s)}(b_{i,k}^{(s)})\le x_k^{(s)}\le x_k^{(s)}(b_{i,k}^{(s)}+1)$. This must be optimized over $\mathbf{x}$ subject to the constraints $\mathbf{G}_1(\boldsymbol{\varkappa})\mathbf{x}^{(s)}=\mathbf{g}_1(\boldsymbol{\varkappa})$ and $\mathbf{G}_2(\boldsymbol{\varkappa})\mathbf{x}^{(s)}=\mathbf{g}_2(\boldsymbol{\varkappa})$ for a given $\boldsymbol{\varkappa}$. Since $\boldsymbol{\varkappa}$ is fixed throughout the optimization $R(\mathbf{r};\boldsymbol{\varkappa})$ is just a constant which can be evaluated and then ignored as far as determination of $\mathbf{x}$ is concerned. We simplify our notation so that $d_1(\mathbf{u})=\left(\mathbf{x}^{(s)}-\boldsymbol{\mu}\right)^t\mathbf{C}_w^{-1}\left(\mathbf{x}^{(s)}-\boldsymbol{\mu}\right)+R(\mathbf{r})$. Similarly,

$$\begin{aligned}\mathbf{x}^{(s)}&=\left(\mathbf{I}-\mathbf{M}^{(s)}(\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{c}(\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\})+\left(\mathbf{I}-\mathbf{M}^{(s)}(\{b_{i,k}^{(s)}\})\right)^{-1}\mathbf{M}^{(b)}(\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}\\&\equiv\mathbf{s}(\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\})+\mathbf{S}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}\end{aligned}$$

where we have defined the n_c-vector $\mathbf{s}=(\mathbf{I}-\mathbf{M}^{(s)})^{-1}\mathbf{c}$ and the $n_c\times n_c'$ matrix $\mathbf{S}=(\mathbf{I}-\mathbf{M}^{(s)})^{-1}\mathbf{M}^{(b)}$.[16] $n_c'<n_c$ is the number of continuous dimensions upon which the supplier response depends. Putting these pieces together we have

$$d_1(\mathbf{x}^{(b)})=\mathbf{x}^{(b)^t}\mathbf{Q}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)}+\mathbf{q}^t\mathbf{x}^{(b)}+const \tag{9}$$

where $\mathbf{Q}$ is an $n_c'\times n_c'$ matrix given by

$$\mathbf{Q}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})=\mathbf{S}^t(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{C}_w^{-1}\mathbf{S}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})$$

[16] In the next section we consider $\mathbf{S}$ further, taking advantage of any redundant variables to simplify the QP task.

$\mathbf{q}$ is an n'-vector given by

$$\mathbf{q}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}) = 2\mathbf{S}^t(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{C}_w^{-1}\big(\mathbf{s}\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}) - \boldsymbol{\mu}\big)^t.$$

and the constant term is

$$const = \big(\mathbf{s}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}) - -\boldsymbol{\mu}\big)^t\mathbf{C}_w^{-1}\big(\mathbf{s}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}) - \boldsymbol{\mu}\big) + R(\mathbf{r})$$

It is easily shown that the matrix $\mathbf{Q}$ is positive semidefinite. $d_1(\mathbf{u}^{(b)})$ must be optimized subject to the constraints

$$\mathbf{G}_1\mathbf{S}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} = \mathbf{g}_1 - \mathbf{G}_1\mathbf{s}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}), \tag{10}$$

$$\mathbf{G}_2\mathbf{S}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} \le \mathbf{g}_2 - \mathbf{G}_2\mathbf{s}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}), \tag{11}$$

$$\underline{\mathbf{x}}^{(s)}(i) \le \mathbf{s}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\}) + \mathbf{S}(\{b_{i,k}^{(s)},\{b_{i,k}^{(b)}\})\mathbf{x}^{(b)} \le \overline{\mathbf{x}}^{(s)}(i)\ \ \forall\, i, \tag{12}$$

$$\underline{\mathbf{x}}^{(b)}(i) \le \mathbf{x}^{(b)} \le \overline{\mathbf{x}}^{(b)}(i)\ \ \forall\, i \tag{13}$$

where $\underline{\mathbf{x}}^{(s)}(i)^t = \begin{bmatrix} x_1(b_{i,1}^{(s)}) & \cdots & x_n(b_{i,n}^{(s)}) \end{bmatrix}$ and $\overline{\mathbf{x}}^{(s)}(i)^t = \begin{bmatrix} x_1(b_{i,1}^{(s)}+1) & \cdots & x_n(b_{i,n}^{(s)}+1) \end{bmatrix}$ are the vectors of breakpoints for the $f_{i,k}^{(s)}$ ($k \in [1,n_c]$) functions. Similarly, for the buyer variables, $\underline{\mathbf{x}}^{(b)}(i)^t = \begin{bmatrix} x_1(b_{i,1}^{(b)}) & \cdots & x_n(b_{i,n}^{(b)}) \end{bmatrix}$ and $\overline{\mathbf{x}}^{(b)}(i)^t = \begin{bmatrix} x_1(b_{i,1}^{(b)}+1) & \cdots & x_n(b_{i,n}^{(b)}+1) \end{bmatrix}$ are the vectors of breakpoints for the $f_{i,k}^{(b)}$ ($k \in [1,n_c]$) functions.

d_1 is optimized in two steps. First we assume a value for the integer variables $\{b_{i,k}^{(s)}\}$ and $\{b_{i,k}^{(b)}\}$ fixing the constraints in Eqs. (10) and (11). We then minimize the quadratic d_1 subject to these constraints using a standard quadratic programming (QP) algorithm[17] to find $\mathbf{x}_{\text{opt}}^{(b)}(\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\})$. This result is then used to find $\mathbf{x}^*\big(\boldsymbol{\varkappa};\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\}\big) = \mathbf{s}\big(\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\}\big) + \mathbf{S}\big(\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\}\big)\mathbf{x}_{\text{opt}}^{(b)}\big(\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\}\big)$. By searching exhaustively over $\{b_{i,k}^{(s)}\}$ and $\{b_{i,k}^{(b)}\}$ we determine

$$\mathbf{x}^{(s)}(\boldsymbol{\varkappa}) = \underset{\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\}}{\arg\min}\ d_1\big(\mathbf{x}^*\big(\boldsymbol{\varkappa};\{b_{i,k}^{(s)}\},\{b_{i,k}^{(b)}\}\big),\boldsymbol{\varkappa}\big)$$

Before turning to the manner in which this optimization is accomplished in section D.3, we discuss a few improvements to the input into the QP algorithm.

D.2 The Quadratic Program

In this section we discuss improvements in the specification of the quadratic programming task to make it better conditioned and more efficient.

It is possible that the matrix $\mathbf{Q}$ used as input to the QP is singular or near singular, depending on the rank of $\mathbf{S}_1$. Consequently, it is important to use an interior point QP algorithm for which this is not a problem.

[17] Numerous third parties supply quadratic programming packages, some based on interior-point methods which are appropriate, in the present context where matrix $\mathbf{Q}$ may be singular.

D.2.1 Variable Elimination

There may be rows or columns of $\mathbf{S}$ which are entirely zero. It is important to take advantage of this structure to eliminate unnecessary computation.

If row i is zero, this indicates that supplier variable $u_i^{(s)}$ is independent of any buyer variable and is simply equal to s_i. If there are any constraints involving only supplier variable i, then these can be checked before calling the QP solver to eliminate wasteful computation.[18] If the constraint is satisfied, we can eliminate it and then call the QP function. If column j of $\mathbf{S}$ is all zero, this means that there is no dependence of $\mathbf{u}^{(s)}$ on $u_j^{(b)}$ and so there is no need to optimize over that buyer variable. We will, however, still have to check that the constraints on $\mathbf{u}^{(s)}$ are satisfied. This observation allows us to simplify the quadratic programming problem by eliminating this unnecessary variable.

We formulate the problem to take advantage of both these observations. Assume there are $m < n_c$ non-zero rows of $\mathbf{S}$ and $m' < n_c'$ non-zero columns. Relabel the rows and columns so that

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{S}_1$ is of size $m \times m'$ and, with a slight abuse of notation, the $\mathbf{0}$'s indicate matrices of the appropriate size. In like fashion we write

$$\mathbf{x}^{(s)} = \begin{bmatrix} \mathbf{x}_1^{(s)} \\ \mathbf{x}_2^{(s)} \end{bmatrix}, \quad \mathbf{x}^{(b)} = \begin{bmatrix} \mathbf{x}_1^{(b)} \\ \mathbf{x}_2^{(b)} \end{bmatrix}$$

where $\mathbf{x}_1^{(s)}$ is an m vector, $\mathbf{x}_2^{(s)}$ is an $n_c - m$ vector, $\mathbf{x}_1^{(b)}$ is an m' vector, and $\mathbf{x}_2^{(b)}$ is an $n_c' - m'$ vector. $\mathbf{x}_1^{(s)}$ indicates those components of $\mathbf{x}^{(s)}$ which vary with buyer variables while $\mathbf{x}_2^{(s)}$ indicates those variables which are constant. $\mathbf{x}_1^{(b)}$ are those buyer variables which affect at least one supplier variable and $\mathbf{x}_2^{(b)}$ are those buyer variables which do not affect any supplier variables. If we also partition $\mathbf{s}$ as $\mathbf{s}^t = \begin{bmatrix} \mathbf{s}_1^t & \mathbf{s}_2^t \end{bmatrix}$ then

$$\mathbf{x}_1^{(s)} = \mathbf{s}_1 + \mathbf{S}_1 \mathbf{x}_1^{(b)} \quad \text{and} \quad \mathbf{x}_2^{(s)} = \mathbf{s}_2.$$

Finally, we will also have need of the partitioned covariance matrix

$$\mathbf{C}^{-1} = \begin{bmatrix} \mathbf{C}_{1,1}^{-1} & \mathbf{C}_{1,2}^{-1} \\ \mathbf{C}_{2,1}^{-1} & \mathbf{C}_{2,2}^{-1} \end{bmatrix}$$

where $(\mathbf{C}_{1,2;w}^{-1})^t = \mathbf{C}_{2,1;w}^{-1}$ since $\mathbf{C}_w^{-1}$ is symmetric.

Including the dependencies on $\boldsymbol{\mu}$, the quadratic form we must optimize is

$$\begin{bmatrix} (\mathbf{s}_1 - \boldsymbol{\mu}_1)^t + (\mathbf{x}_1^{(b)})^t \mathbf{S}_1^t & (\mathbf{s}_2 - \boldsymbol{\mu}_2)^t \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 \mathbf{C}_{1,1}^{-1} \mathbf{W}_1 & \mathbf{W}_1 \mathbf{C}_{1,2}^{-1} \mathbf{W}_2 \\ \mathbf{W}_2 \mathbf{C}_{2,1}^{-1} \mathbf{W}_1 & \mathbf{W}_2 \mathbf{C}_{2,2}^{-1} \mathbf{W}_2 \end{bmatrix} \begin{bmatrix} \mathbf{s}_1 - \boldsymbol{\mu}_1 + \mathbf{S}_1 \mathbf{u}_1^{(b)} \\ \mathbf{s}_2 - \boldsymbol{\mu}_2 \end{bmatrix}$$

[18] Additionally, the QP doesn't like to have a row of zeros in the constraint matrix.

where we have made the obvious definitions

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 \end{bmatrix} \quad \text{and} \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}.$$

Expanding the quadratic, the constant term is

$$(\mathbf{s}_1 - \mu_1)^t \mathbf{W}_1 \mathbf{C}_{1,1}^{-1} \mathbf{W}_1 (\mathbf{s}_1 - \mu_1) + (\mathbf{s}_2 - \mu_2)^t \mathbf{W}_2 \mathbf{C}_{2,2}^{-1} \mathbf{W}_2 (\mathbf{s}_2 - \mu_2) + 2(\mathbf{s}_1 - \mu_1)^t \mathbf{W}_1 \mathbf{C}_{1,2}^{-1} \mathbf{W}_2 (\mathbf{s}_2 - \mu_2).$$

The linear term is

$$\mathbf{q} = \left((\mathbf{s}_1 - \mu_1)^t \mathbf{W}_1 \mathbf{C}_{1,1}^{-1} \mathbf{W}_1 \mathbf{S}_1^t + 2(\mathbf{s}_2 - \mu_2)^t \mathbf{W}_2 \mathbf{C}_{2,1}^{-1} \mathbf{W}_1 \mathbf{S}_1\right) \mathbf{x}_1^{(b)}$$

and the quadratic term is

$$\mathbf{Q} = (\mathbf{x}_1^{(b)})^t \mathbf{S}_1^t \mathbf{W}_1 \mathbf{C}_{1,1}^{-1} \mathbf{W}_1 \mathbf{S}_1 \mathbf{x}_1^{(b)}.$$

The constraints involving the optimization variables $\mathbf{u}_1^{(b)}$ are

$$\begin{aligned} \mathbf{G}_{1,1} \mathbf{S}_1 \mathbf{x}_1^{(b)} &= \mathbf{g}_1 - \mathbf{G}_{1,1} \mathbf{s}_1 - \mathbf{G}_{1;2} \mathbf{s}_2, \\ \mathbf{G}_{2;1} \mathbf{S}_1 \mathbf{x}_1^{(b)} &\leq \mathbf{g}_2 - \mathbf{G}_{2,1} \mathbf{s}_1 - \mathbf{G}_{2;2} \mathbf{s}_2, \\ \underline{\mathbf{x}}_1^{(s)}(i) \leq \mathbf{s}_1 + \mathbf{S}_1 \mathbf{x}_1^{(b)} &\leq \overline{\mathbf{x}}_1^{(s)}(i) \ \ \forall\, i, \\ \underline{\mathbf{x}}_1^{(b)}(i) \leq \mathbf{x}_1^{(b)} &\leq \overline{\mathbf{x}}_1^{(b)}(i) \ \ \forall\, i \end{aligned}$$

where we have simplified notation by dropping the dependence on breakpoints and defined

$$\mathbf{G}_i = \begin{bmatrix} \mathbf{G}_{i;1} & \mathbf{G}_{i;2} \end{bmatrix}$$

i.e., $\mathbf{G}_{i;1}$ is the columns of $\mathbf{G}_1$ involving the active buyer variables while $\mathbf{G}_{i;2}$ is formed from the inactive buyer variables. Similarly, the constraints which can be checked prior to even solving the QP are

$$\underline{\mathbf{x}}_2^{(s)}(i) \leq \mathbf{s}_2 \leq \overline{\mathbf{x}}_2^{(s)}(i) \ \ \forall\, i.$$

D.2.2 QP Normalization

There are two types of normalizations we perform for increased numerical stability. If the scales of the continuous dimensions vary greatly (i.e., one dimension has an ideal value of 10^{-2} and another dimension has ideal value 10^6), then the matrix $\mathbf{C}^{-1}$ becomes badly conditioned, which makes $\mathbf{S}$ badly conditioned and thus makes the QP unstable. Similarly, if the breakpoints have radically, different lengths, the QP problem can also become unstable. We consider how to remedy each of these problems.

Dimension normalization

The quadratic form to be minimized is $(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu})$. We modify this problem by defining the dimensionless variables ϕ_i through $x_i = \phi_i\mu_i$ in terms of which

$$(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{C}^{-1}(\mathbf{x}-\boldsymbol{\mu}) = (\boldsymbol{\phi}-\mathbf{1})^t\boldsymbol{\mathcal{C}}^{-1}(\boldsymbol{\phi}-\mathbf{1})$$

where $\boldsymbol{\mathcal{C}}^{-1} = \mathbf{D}\mathbf{C}^{-1}\mathbf{D}$ and $\mathbf{D} = \mathrm{diag}(\boldsymbol{\mu})$. When making this variable transformation we must assure that the constraints on $\mathbf{x}$ are adjusted so that $\mathbf{G}_1\mathbf{x} = \mathbf{g}_1$ and $\mathbf{G}_2\mathbf{x} \leq \mathbf{g}_2$ become the constraints $\boldsymbol{\mathcal{G}}_1\boldsymbol{\phi} = \mathbf{g}_1$ and $\boldsymbol{\mathcal{G}}_2\boldsymbol{\phi} \leq \mathbf{g}_2$ on $\boldsymbol{\phi}$ where $\boldsymbol{\mathcal{G}}_{1,2} = \mathbf{G}_{1,2}\mathbf{D}$. If there are global bounds of the form $\underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}}$, then these are translated to $\underline{\boldsymbol{\phi}} \leq \boldsymbol{\phi} \leq \overline{\boldsymbol{\phi}}$ where $\underline{\boldsymbol{\phi}} = \mathbf{D}^{-1}\underline{\mathbf{u}}$ and $\overline{\boldsymbol{\phi}} = \mathbf{D}^{-1}\overline{\mathbf{u}}$.

In addition to these changes, we also need to modify the capabilities appropriately. Recall Eq. (2) which reads (after dropping the dependence on the discrete variables)

$$x_i^{(s)} = \sum_k f_{i,k}^{(s)}(x_k^{(s)}) + f_{i,k}^{(b)}(x_k^{(b)}). \tag{14}$$

In terms of the dimensionless variables we have

$$\phi_i^{(s)} = \sum_k \frac{f_{i,k}^{(s)}(\phi_k^{(s)}\mu_k)}{\mu_i} + \frac{f_{i,k}^{(b)}(\phi_k^{(b)}\mu_k)}{\mu_i}. \tag{15}$$

Thus, when we use the dimensionless variables, the function $f_{i,k}$ needs to be modified so that the y dimension is divided by μ_i and the x dimension is multiplied by μ_k.

Once the optimization has determined an optimal $\boldsymbol{\phi}$, we return to the original variables through $\mathbf{x} = \mathbf{D}\boldsymbol{\phi}$.

Breakpoint range normalization

For larger problems it is sometimes important to normalize the input variables for the QP solver. This is carried out in order to improve the QP search performance in situations where there is large variation in the range of the segments $\phi_k^{(b)}(b_k) \leq \phi_k^{(b)} \leq \phi_k^{(b)}(b_k+1)$ and the magnitude of their values. This can be accomplished by introducing the normalization variables $\hat{\phi}_k^{(b)} = (\phi_k^{(b)} - \phi_k^{(b)}(b_k))/(\phi_k^{(b)}(b_k+1) - \phi_k^{(b)}(b_k))$. Here $\hat{\phi}_k^{(b)}$ will have minimum value of zero and maximum of one. We can now express $\boldsymbol{\phi}^{(b)}$ as

$$\boldsymbol{\phi}^{(b)} = \boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \mathbf{p} \tag{16}$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix $\boldsymbol{\Lambda} = \mathrm{diag}(\phi_1^{(b_1+1)} - \phi_1^{(b_1)}, \phi_2^{(b_2+1)} - \phi_2^{(b_2)}, \cdots, \phi_m^{(b_m+1)} - \phi_m^{(b_m)})$ and $\mathbf{p}^t = [\phi_1^{(b_1)}, \phi_2^{(b_2)}, \cdots, \phi_m^{(b_m)}]$.

Expanding Eq.(9) using the definition of $\boldsymbol{\phi}$ in Eq.(16), we obtain

$$\begin{aligned} d_1(\mathbf{w};\{b_{i,k}\}) =& \frac{1}{2}\hat{\boldsymbol{\phi}}^{(b)t}\boldsymbol{\Lambda}\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \mathbf{q}^t(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{b} + \mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\boldsymbol{\Lambda}\hat{\boldsymbol{\phi}}^{(b)} + \\ & \frac{1}{2}\mathbf{p}^t\mathbf{D}\mathbf{Q}(\{b_{i,k}\})\mathbf{D}\mathbf{p} + \mathbf{q}^t(\{b_{i,k}\})\mathbf{D}\mathbf{p}. \end{aligned}$$

Hence the quadratic programming problem to be solved in terms of $\hat{\boldsymbol{\phi}}$ is

$$QP(\hat{\mathbf{x}}^{(b)};\{b_{i,k}\}) = \frac{1}{2}\hat{\mathbf{x}}^{(b)t}\hat{\mathbf{Q}}(\{b_{i,k}\})\hat{\mathbf{x}}^{(b)} + \hat{\mathbf{q}}^t(\{b_{i,k}\})\hat{\mathbf{u}}^{(b)}), \tag{17}$$

where $\hat{\mathbf{Q}} = \mathbf{\Lambda D Q}(\{b_{i,k}\})\mathbf{D\Lambda}$ and $\hat{\mathbf{q}}^t = (\mathbf{q}^t(\{b_{i,k}\}) + \mathbf{p}^t\mathbf{DQ}(\{b_{i,k}\}))\mathbf{D\Lambda}$. Similarly, the equality and inequality constraints of Eqs. (10) and (11) expressed in terms of $\hat{\boldsymbol{\phi}}$ are

$$\hat{\mathbf{G}}_1(\{b_{i,k}\})\hat{\boldsymbol{\phi}}^{(b)} = \hat{\mathbf{g}}_1 \quad \text{and} \quad \hat{\mathbf{G}}_2(\{b_{i,k}\})\hat{\boldsymbol{\phi}}^{(b)} \leq \hat{\mathbf{g}}_2, \tag{18}$$

where

$$\begin{aligned}
\hat{\mathbf{G}}_1(\{b_{i,k}\}) &= \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{D\Lambda}, \\
\hat{\mathbf{G}}_2(\{b_{i,k}\}) &= \mathbf{G}_2\mathbf{S}(\{b_{i,k}\})\mathbf{D\Lambda}, \\
\hat{\mathbf{g}}_1 &= \mathbf{g}_1 - \mathbf{G}_1\mathbf{s}(\{b_{i,k}\}) - \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{Dp}, \\
\hat{\mathbf{g}}_2 &= \mathbf{g}_2 - \mathbf{G}_2\mathbf{s}(\{b_{i,k}\}) - \mathbf{G}_1\mathbf{S}(\{b_{i,k}\})\mathbf{Dp}.
\end{aligned}$$

D.3 Optimization Over The Breakpoint Variables

We have detailed how to determine $\mathbf{x}^{(b)}_{\text{opt}}(\boldsymbol{\varkappa}, \{b^{(s)}_{i,k}\}, \{b^{(b)}_{i,k}\})$ given a specification of all $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$, but how do we determine the best feasible choice of $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$? Before we answer that question, we have to examine the infeasibilities that might arise due to improper choices of $\{b^{(s)}_{i,k}\}$ and $\{b^{(b)}_{i,k}\}$.

D.3.1 Elimination of $\{b_{i,k}\}$ Infeasibilities

Most of the settings of $\{b^{(b)}_{i,k}\}$ or $\{b^{(s)}_{i,k}\}$ will not yield a feasible solution because Eq. (13) or Eq. (12) will be violated. The reason is that, for any k, it is highly likely to get two segments $x^{(b)}_k(b_{i,k}) \leq x^{(b)}_k \leq x^{(b)}_k(b_{i,k}+1)$ and $x^{(b)}_k(b_{j,k}) \leq x^{(b)}_k \leq x^{(b)}_k(b_{j,k}+1)$ that do not overlap (i.e., the optimizer will not be able to find a $x^{(b)}_k$ value that satisfies both bounds, and hence no feasible solution will be found). The same is also true for supplier response segment $x^{(s)}_k$.

To eliminate those infeasibilities, we construct a super set of breakpoints $\{\varpi^b_k\}$ such that

$$\varpi^{(b)}_k = \bigcup_{i=1}^{n} \{x^{(b)}_k(b^{(b)}_{i,k} = 1), x^{(b)}_k(b^{(s)}_{i,k} = 2), \cdots, x^{(b)}_k(b^{(b)}_{i,k} = \kappa^{(b)}_{i,k})\}$$

Hence each piecewise linear function $f_{i,k}$ will be described by a set of breakpoints $\varpi_k = \{x_k(1), \cdots, x_k(\kappa_k)\}$ and function values $f_{i,k}(x_k(1)), \cdots, f_{i,k}(x_k(\kappa_k))$. These function values at the new breakpoints are linearly interpolated from the old breakpoint-function pairs. After this reduction we have $\kappa_k - 1$ different $b_k \in [1, \kappa_k - 1]$ values, which eliminates the need for constraint Eq. (13).

The continuous supplier variables and constraint Eq. (12) are more problematic. Suppressing any dependence on $\boldsymbol{\varkappa}$, we know

$$x^{(s)}_i = c_i(\{b^{(s)}_{i,1}, \cdots, b^{(s)}_{i,i-1}\}, \{b^{(b)}_{i,k}\}) + \sum_{k<i} m^{(s)}_{i,k}(b^{(s)}_{i,k})x^{(s)}_k + \sum_k m^{(b)}_{i,k}(b^{(b)}_{i,k})x^{(b)}_k.$$

If $\{b_{i,k}^{(b)}\}$ and $\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}$ are known, then this expression allows us to bound the values $x_i^{(s)}$ may take on, and thus bound the interval in which the feasible $b_{i,i}^{(s)}$ lie. If we define the functions

$$\text{smax}(s,x,y) = \begin{cases} s\max(x,y) & \text{if } s \geq 0 \\ s\min(x,y) & \text{if } s < 0 \end{cases} \quad \text{and} \quad \text{smin}(s,x,y) = \begin{cases} s\min(x,y) & \text{if } s \geq 0 \\ s\max(x,y) & \text{if } s < 0 \end{cases}$$

then the minimum and maximum values that $x_i^{(s)}$ may take on are:

$$\begin{aligned} \min(x_i^{(s)}) = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \sum_{k<i} \text{smin}\big(m_{i,k}^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}+1)\big) \\ + \sum_k \text{smin}\big(m_{i,k}^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}+1)\big), \end{aligned} \tag{19}$$

and

$$\begin{aligned} \max(x_i^{(s)}) = c_i(\{b_{i,1}^{(s)}, \cdots, b_{i,i-1}^{(s)}\}, \{b_{i,k}^{(b)}\}) + \sum_{k<i} \text{smax}\big(m_{i,k}^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}), x_k^{(s)}(b_{i,k}^{(s)}+1)\big) \\ + \sum_k \text{smax}\big(m_{i,k}^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}), x_k^{(b)}(b_{i,k}^{(b)}+1)\big) \end{aligned} \tag{20}$$

respectively. The above equations imply that we can fully translate a complete specification of buyer variable breakpoint intervals to a corresponding set of feasible supplier variable breakpoint intervals. We will search over the breakpoints using a branch-and-bound algorithm, and will need to be able to infer the restriction on the values of feasible supplier breakpoint intervals given a partial specification of buyer breakpoint interval variables. This inference, which narrows the domains of the supplier breakpoint variables, is called constraint propagation. Before describing how the branch-and-bound algorithm search over breakpoint interval variables is accomplished, we show how the constraint propagation may be obtained.

Narrowing of Supplier Breakpoint Domains Through Propagation

Let $\boldsymbol{\beta}_k^{(b)}$ be a vector of integers specifying a set of breakpoint intervals for the kth buyer variable. $|\boldsymbol{\beta}_k| = 1$ if the interval is uniquely determined, and $|\boldsymbol{\beta}_k| = \kappa_k^{(b)} - 1$ if no restrictions on the interval have been specified. We index the lth component of $\boldsymbol{\beta}_k^{(b)}$ as $\beta_{k;l}^{(b)}$, $l \in [1, \kappa_k^{(b)} - 1]$. A set of breakpoints for all buyer dimensions is written as $\{\boldsymbol{\beta}_k^{(b)}\}$. In analogous fashion, we define $\boldsymbol{\beta}_k^{(s)}$, $\beta_{k;l}^{(s)}$, $l \in [1, \kappa_k^{(s)} - 1]$, and $\{\boldsymbol{\beta}_k\}$ for the supplier breakpoint interval variables.

From a specification $\{\boldsymbol{\beta}_k^{(b)}\}$ we infer the feasible domains $\{\boldsymbol{\beta}_k^{(s)}\}$ using $\max(x_i^{(s)})$ and $\min(x_i^{(s)})$ where:

$$\begin{aligned} \max(x_i^{(s)}) = \sum_k \max_{b_k^{(b)} \in \boldsymbol{\beta}_k^{(b)}} \Big\{ c_{i,k}(\{b_k^{(b)}\}) + \text{smax}\big(m_{i,k}^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}+1)\big) \Big\} + \\ \sum_{k<i} \max_{b_k^{(s)} \in \boldsymbol{\beta}_k^{(s)}} \Big\{ c_{i,k}(\{b_k^{(s)}\}) + \text{smax}\big(m_{i,k}^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}+1)\big) \Big\} \end{aligned} \tag{21}$$

and

$$\min(x_i^{(s)}) = \sum_k \min_{b_k^{(b)} \in \boldsymbol{\beta}_k^{(b)}} \left\{ c_{i,k}(\{b_k^{(b)}\}) + \text{smin}\big(m_{i,k}^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}), x_k^{(b)}(b_k^{(b)}+1)\big) \right\} +$$
$$\sum_{k<i} \min_{b_k^{(s)} \in \boldsymbol{\beta}_k^{(s)}} \left\{ c_{i,k}(\{b_k^{(s)}\}) + \text{smin}\big(m_{i,k}^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}), x_k^{(s)}(b_k^{(s)}+1)\big) \right\} \tag{22}$$

We begin with $i = 1$ (where no supplier variables contribute to $x_1^{(s)}$) and determine the interval $I_1 = [\min(x_1^{(s)}), \max(x_1^{(s)})]$. We then form the vector $\boldsymbol{\beta}_1^{(s)}$ with components $\{\beta_{1;l}^{(s)} | [x_1^{(s)}(\beta_{1;l}), x_1^{(s)}(\beta_{1;l}+1)] \cap I_1 \neq \emptyset\}$. Equations (21) and (22) can then be iterated to find the interval $I_2 = [\min(x_2^{(s)}), \max(x_2^{(s)})]$ using the previously calculated $\min(x_1^{(s)})$ and $\max(x_1^{(s)})$. With I_2 known, the components of $\boldsymbol{\beta}_2^{(s)}$ are determined as above. Iterating this procedure, we can find the potentially narrowed domains of all $\{\beta_k^{(s)}\}$.

We now return to the optimization problem: we need to search over all $\prod_k \kappa_k^{(b)} \times \prod_k \kappa_k^{(s)}$ combinations of breakpoints to determine the combination that minimizes $d_1\left(\mathbf{x}_{\text{opt}}(\boldsymbol{\varkappa}, \{b_k^{(b)}\}, \{b_k^{(s)}\}), \boldsymbol{\varkappa}\right)$. The simplest manner in which this accomplished is via exhaustive enumeration over all $\{b_k^{(b)}\}$ and $\{b_k^{(s)}\}$. However, once all $b_k^{(b)}$ have been specified, we need only search exhaustively over the corresponding $\{\beta_k(s)\}$ inferred as above. There are, however, many ways to speed up the search over the buyer breakpoint intervals. We describe two methods.

D.3.2 Local Search

This method is extremely fast but not guaranteed to return the global optimum across all buyer breakpoint intervals. Here, initial random and feasible[19] $\{b_k^{(s)}\}$, and $\{b_k^{(b)}\}$ sets are first created. After the continuous optimization is carried out using the initial $\{b_k^{(s)}\}$, and $\{b_k^{(b)}\}$, a new $\{\tilde{b}_k^{(s)}\}$, and $\{\tilde{b}_k^{(b)}\}$ set is generated from the old one by examining whether each generated $x_k^{(b)}(\{b_k^{(s)}\}, \{b_k^{(b)}\})$ solution is on the boundaries of the interval $x_k^{(b)}(b_k) \leq x_k^{(b)} \leq x_k^{(b)}(b_k+1)$ or $x_k^{(s)}(b_k) \leq x_k^{(s)} \leq x_k^{(s)}(b_k+1)$. If $x_k^{(b)}$ is on the lower bound of the interval, $\tilde{b}_k^{(b)} = b_k^{(b)} - 1$; similarly, if it is on the upper bound, $\tilde{b}_k^{(b)} = b_k^{(b)} + 1$. If $x_k^{(b)}$ is not on either end, then $\tilde{b}_k^{(b)} = b_k^{(b)}$. Of course this done while maintaining $\tilde{b}_k^{(b)}$ in the interval $1 \leq \tilde{b}_k^{(b)} \leq \kappa_k^{(b)} - 1$. Supplier breakpoint variables are updated in a similar manner. The procedure is repeated until no decrease in distance is attained.

D.3.3 Branch-and-Bound Search

Branch-and-bound is an efficient procedure used to exhaustively search a space and eliminate configurations that can be proven to be suboptimal. The theory behind branch-and-bound is well understood [5]. The idea is to structure the space as a tree of partially-completed possibilities and to explore the tree so that good complete solutions are constructed early on. Branches of the tree that can be shown to yield solutions

[19] Feasible in the sense that they do not violate the constraints in equations (10), (11), (12), and (13).

which cannot be better than the currently best solution need not be explored. Heuristics indicate the order in which to explore the tree and provide a lower bound on the fitness of partially completed solutions for pruning of suboptimal branches..

In the present context, we carry out branch-and-bound search over the space of possible buyer breakpoint combinations. The tree is structured beginning from a node at which none of the $b_k^{(b)}$ have been assigned. From this node, the tree branches into the $\kappa_i^{(b)} - 1$ possible assignations for $b_i^{(b)}$. From any of these nodes, the tree branches into the $\kappa_j^{(b)}$ possible values for $b_j^{(b)}$. The tree branches at each node to assign the remaining variables. At the leaves of the tree, all buyer breakpoint variables have been assigned.

From any node, the search proceeds by choosing the most promising node to consider, as indicated by a variable ordering heuristic and considering the possible values for that variable in the order indicated by the value ordering heuristic. A complete solution and its accompanying distance is obtained when the search ends at a leaf of the tree. To avoid unnecessary exploration, a lower bound heuristic indicates when a partially completed solution (node) need not be expanded because all completions will have higher distance than the best located so far. We describe these three heuristics for search over buyer breakpoints.

Lower Bound

Given a set of buyer breakpoint interval vectors $\{\boldsymbol{\beta}_k^{(b)}\}$, we can propagate to $\{\boldsymbol{\beta}^{(s)}\}$ and the intervals $[\min(x_i^{(s)}), \max(x_i^{(s)})]$. A lower bound on the value d_1 can be obtained by ignoring the supplier capabilities and minimizing

$$d_1(\mathbf{x}) = \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right)^t \mathbf{C}_w^{-1} \left(\mathbf{x}^{(s)} - \boldsymbol{\mu}\right) \tag{23}$$

subject to

$$\mathbf{G}_1 \mathbf{x}^{(\mathbf{s})} = \mathbf{g}_1 \tag{24}$$

$$\mathbf{G}_2 \mathbf{x}^{(\mathbf{s})} \leq \mathbf{g}_2 \tag{25}$$

$$\min(x_i^{(s)}) \leq x_i^{(s)} \leq \max(x_i^{(s)}), \tag{26}$$

The lower bound thus obtained is denoted by $\mathrm{lb}(\{\boldsymbol{\beta}^{(b)}\})$.

Variable and Value Ordering

The lower bound $\mathrm{lb}(\{\boldsymbol{\beta}^{(b)}\})$ can be used to define an ordering over variables and the values any variable may take on. We define the functions $L_{k,l}$ which operate on the set $\{\boldsymbol{\beta}^{(b)}\}$ by locking the kth buyer breakpoint variable to $\beta_{k;l}$. At any node we can determine the lower bounds $\mathrm{lb}\left(L_{k,l}\{\boldsymbol{\beta}^{(b)}\}\right)$. We can select to explore the variables in order of increasing lb_k where

$$\mathrm{lb}_k(\{\boldsymbol{\beta}^{(b)}\}) = \frac{\sum_{l=1}^{|\boldsymbol{\beta}_k^{(b)}|} \mathrm{lb}\left(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\right)}{\sum_{l=1}^{|\boldsymbol{\beta}_k^{(b)}|} 1},$$

i.e., the average over all possible completions for $b_k^{(b)}$.[20] Within any given variable the (say k^*) values can be explored in order of increasing $\mathrm{lb}\big(L_{k^*,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$. If we are willing to do more computation, we can explore the branches in an interleaved order according to all the $\mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$.

Pruning and Search

During the branch-and-bound search, branches can be pruned if the node has $\mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$ greater than the distance of the lowest examined leaf node.

Search begins at the root node where each $\boldsymbol{\beta}_k^{(b)} = [1, \cdots, \kappa_k^{(b)} - 1]$. We applying the ordering heuristics above to move to a node where one breakpoint variable has been fixed. From that node, we move down the tree, fixing a value at a time until we reach a leaf. We then progressively backtrack up the tree, eliminating branches whose lower bounds are greater than the best leaf examined.

Once the optimal set of buyer breakpoints has been determined, we search exhaustively over the supplier breakpoints feasible with this choice according to Equations (19) and (20).

Improvements

We have not yet exploited the fact that, from any node having $\{\boldsymbol{\beta}_k^{(b)}\}$, the constraints in Equation (12) may narrow the domains of the buyer breakpoint variables deeper in the tree. The computational cost of narrowing the breakpoint domains may be warranted if the pruning thus obtained is sufficient.

E Phase 2 Optimization

In this section we detail an efficient algorithm to optimize Eq. (5):

$$\arg\min_{\boldsymbol{\varkappa}} \; d_1(\mathbf{x}(\boldsymbol{\varkappa}), \boldsymbol{\varkappa}) + d_2(\boldsymbol{\varkappa}) \quad \text{subject to} \quad \mathcal{C}^{(b)}(\boldsymbol{\varkappa}) \wedge \mathcal{C}^{(s)}(\boldsymbol{\varkappa}).$$

Our approach is a marriage of constraint programming and local search.

E.1 Local Search over D-neighborhoods

We begin with a definition. The d-neighbors of a discrete configuration $\boldsymbol{\varkappa}$ are all those configurations, $\boldsymbol{\varkappa}'$, which differ from $\boldsymbol{\varkappa}$ in d attributes. Attribute i differs in two configurations if $\varkappa_i \neq \varkappa_i'$. Thus the distance varies from 0 if the configurations are identical, up to n_d if all attributes are different. The D-neighborhood of $\boldsymbol{\varkappa}$ is the union of all $d = 1$ neighbors, $d = 2$ neighbors, up to $d = D$ neighbors. The D-neighborhood around $\boldsymbol{\varkappa}$ is indicated by $\mathcal{N}_D(\boldsymbol{\varkappa})$. We note that, for any configuration, its n_d neighborhood is the entire discrete space.

Using the D-neighborhood, we apply a standard hillclimbing algorithm which respects all the discrete constraints. D is specified before the algorithm begins and serves as an efficient manner to trade off running

[20] We might also take the minimum over all possible completions $\mathrm{lb}_k(\{\boldsymbol{\beta}^{(b)}\}) = \min_{l \in [1, |\boldsymbol{\beta}_k^{(b)}|]} \mathrm{lb}\big(L_{k,l}(\{\boldsymbol{\beta}^{(b)}\})\big)$.

1	***11
2	**1*1
3	**11*
4	*1**1
5	*1*1*
6	*11**
7	1***1
8	1**1*
9	1*1**
10	11***

(a)

1	***+1
2	**1*1
3	**1+*
4	*+**1
5	*+*+*
6	*+1**
7	1***1
8	1**+*
9	1*1**
10	1+***

(b)

Table 5: Configurations examined in the $D = 3$ neighborhood search around (a) the initial configuration 11111 and (b) the initial configuration 0+0+0 (duplicate search configurations are indicated in bold) .

time with the quality of the final solution. The hillclimbing algorithm takes as input a starting configuration $\boldsymbol{x}_t$ and finds the configuration $\boldsymbol{x}_{t+1} \in \mathcal{N}_D(\boldsymbol{x}_t)$ which minimizes $d_1(\mathrm{x}(\boldsymbol{x}), \boldsymbol{x}) + d_2(\boldsymbol{x})$ subject to $\mathcal{C}^{(b)}(\boldsymbol{x}) \wedge \mathcal{C}^{(s)}(\boldsymbol{x})$. The algorithm then iterates again starting from $\boldsymbol{x}_{t+1}$. The hillclimbing algorithm terminates when it finds a configuration $\boldsymbol{x}_T$ which has a distance less than or equal to all of its D-neighbors. If $D = n_d$, this is the global minimum. Given a function which exhaustively searches a D-neighborhood for the lowest distance feasible configuration, a D-neighborhood hillclimbing algorithm is easily implemented.

There is one noteworthy point about the implementation of an efficient D-neighborhood hillclimber. Consider a $D = 3$ search over $n_d = 5$ discrete variables. Without loss of generality, we label the initial configuration as 11111. In the $D = 3$ neighborhood search, the configurations examined are given in Table 5(a) as indicated by the *s. Let us assume that the best neighbor in the D neighborhood of 11111 differs in components 2 and 4. We indicate this best neighbor as 1+1+1, which serves as the starting configuration for a new $D = 3$ search. The configurations examined in the second neighborhood search are given in Table 5(b). From Table 5(b) we see that the search branches 2, 7, and 10 are redundant, having already been performed in the D neighborhood search around 11111.[21] Thus these branches may be pruned in the second D neighborhood search. A search branch covers a previous search and may be pruned if the vector of positions of the wildcard characters covers the vector of positions at which the previous best solution was found. The binary relation covers is defined as

$$\boldsymbol{x} \text{ covers } \boldsymbol{x}' \iff x'_i \in \boldsymbol{x} \ \forall \ x'_i \in \boldsymbol{x}'.$$

[21]The duplication of search effort is most acute when D is large and the fittest neighbor is found very close by (small d).

The covers relation is transitive but not reflexive. Thus, to increase the efficiency, we can keep a list of the previous branches searched and ignored search branches that have already been covered.

E.1.1 Improvements

Throughout the present discussion we have assumed that D remains constant through the hillclimbing process. There are good reasons why this should not be the case. With any reasonable landscape correlation structure, we know that lower distance neighbors are more likely to be found early in the hillclimbing process. Thus a reasonable heuristic is to start with a larger D early on and gradually lower it with time.

E.2 Exhaustive Search of a D-neighborhood

To complete the discussion of the phase 2 optimization, we describe a computationally efficient way in which to find the lowest distance feasible configuration within a D-neighborhood. We use a branch-and-bound method with heuristics tuned for the present problem. The factors tuned include the variable order in which we search, the order of the values any given variable may take, and a heuristic which says when certain branches of the search space may be safely ignored.

We use heuristics which early on in the search generate low-distance configurations, and later prune parts of the search that we can prove must have higher distance than the best configuration generated so far.

E.2.1 Heuristics And Lower Bounds

We focus on obtaining good heuristics for the discrete contribution to the total distance, then later incorporate the continuous and range contributions since these are simpler.

At any point in the search, some variables have been assigned and some have not. We divide up the sum over the $Z_{i,j}(\varkappa_i, \varkappa_j)$ into four contributions

$$\begin{aligned} Q_d Z_w(\boldsymbol{\varkappa}) = {} & Q_d \sum_{i \text{ locked}} \sum_{j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) + Q_d \sum_{i \text{ locked}} \sum_{j \text{ floating}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) \\ & + Q_d \sum_{i \text{ floating}} \sum_{j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) + Q_d \sum_{i \text{ floating}} \sum_{j \text{ floating}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) \\ \equiv {} & Z_{\mathrm{l,l}} + Z_{\mathrm{l,f}} + Z_{\mathrm{f,l}} + Z_{\mathrm{f,f}} \end{aligned}$$

Since we are minimizing and all $Z_{i,j}(\varkappa_i, \varkappa_j)$, w_i, and Q_d are positive, we can prune any search node which has $Z_{\mathrm{l,l}} >$ than the distance of the lowest distance configuration found thus far, since additional terms will only add to the sum.

We obtain a lower bound on the distance by lower-bounding each of the above four terms. Just taking the lowest value in the tables can yield differing values for any given variable, e.g., the values for a single

variable $\varkappa_i$:

$$\varkappa_i^* = \arg\min_{\varkappa_i} Z_{i,j}(\varkappa_i, \varkappa_j), \quad \varkappa_i^* = \arg\min_{\varkappa_i} Z_{j,i}(\varkappa_j, \varkappa_i), \quad \text{and} \quad \varkappa_i^* = \arg\min_{\varkappa_i} Z_{i,i}(\varkappa_i)$$

may all be different. To correct this we write

$$Q_d Z_w = Z_{\mathrm{l,l}} + Z_{\mathrm{f,f}} + \sum_{i \text{ floating}} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$$

where $Z_{\mathrm{l,l}}$ and $Z_{\mathrm{f,f}}$ are calculated as before and

$$Z_{\mathrm{l,f}}^{(i)}(\varkappa_i) = Q_d\Big(w_i^2 Z_{i,i}(\varkappa_i) + w_i \sum_{j \text{ locked}} w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)\Big).$$

The best choice for the floating variable $\varkappa_i$ is

$$\varkappa_i^* = \arg\min_{\varkappa_i} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i). \tag{27}$$

In this way all the possible choices for $\varkappa_i$ are at least consistent, and the true lower limit is still guaranteed to be higher than this (since the $Z_{\mathrm{f,f}}$ contribution can only be higher).

This approach also naturally suggests a variable ordering and a value ordering within any variable. We can try the variables in increasing order of their corresponding $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$ so that variable i, which has the smallest $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$, is tried first. For any given variable $\varkappa_i$ we can also order the possible value assignations for $\varkappa_i$ according to the array of values $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$ so that the value $\varkappa_i = \varkappa_i^*$ is tried first and the value which maximizes $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$ is tried last.

With little extra work we can impose a consistency requirement raising the lower bound for $Z_{\mathrm{f,f}}$. We decompose the sum as follows

$$Z_{\mathrm{f,f}} \equiv Q_d \sum_{i,j \text{ floating}: j \neq i} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j) = Q_d \sum_{i,j \text{ floating}: j < i} w_i w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big).$$

Instead of taking the minimum component-wise as previously suggested, *i.e* instead of

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}: j \neq i} \min_{\varkappa_i, \varkappa_j} \big[w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j)\big],$$

we take the minimum so that

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}: j < i} \min \big[w_i w_j \big(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\big)\big]. \tag{28}$$

which enforces pairwise consistency between variables i and j.

The lower bound, lb, we have obtained is thus

$$\mathrm{lb} = Z_{\mathrm{l,l}} + Z_{\mathrm{f,f}}^* + \sum_i Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*)$$

where $\varkappa_i^*$ is determined from Eq. (27) and $Z_{\mathrm{f,f}}^*$ is determined from Eq. (28).

Integration Including Continuous and Range Contributions

In this section we summarize the complete heuristic specification, including the effects of the continuous and range variables. As above, we use the definitions

$$Z_{\mathrm{l,l}} = Q_d \sum_{i,j \text{ locked}} w_i w_j Z_{i,j}(\varkappa_i, \varkappa_j)$$

$$Z_{\mathrm{l,f}}^{(i)}(\varkappa_i) = Q_d w_i \sum_{j \text{ locked}} w_j \left(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_j, \varkappa_i)\right)$$

$$Z_{\mathrm{f,f}}^* = Q_d \sum_{i,j \text{ floating}, j<i} \min\left[w_i w_j \left(Z_{i,j}(\varkappa_i, \varkappa_j) + Z_{j,i}(\varkappa_i, \varkappa_j)\right)\right]$$

so that a lower bound to the complete discrete contribution to the distance is $Z_{\mathrm{l,l}} + Z_{\mathrm{f,f}}^* + \sum_i Z_{\mathrm{l,f}}^*(\varkappa_i^*)$ where $\varkappa_i^* = \arg\min_{\varkappa_i} Z_{\mathrm{l,f}}^{(i)}(\varkappa_i)$.

We now extend this result to include the continuous and range variables. The continuous and range variables depend on the discrete variables through the mask. Since we expect there will be relatively few masks we exhaustively enumerate all the discrete possibilities. Consequently, we define $d_1(\boldsymbol{\varkappa}_{\text{floating}}|\boldsymbol{\varkappa}_{\text{locked}})$ as the distance contribution made by the continuous and range variables (the d_1 contribution to Eq. (5)). We define the continuous + range contribution for any given floating variable i as

$$d_1^{(i)}(\varkappa_i) \equiv \min_{\text{floating}\setminus i} d_1(\boldsymbol{\varkappa}_{\text{floating}}|\boldsymbol{\varkappa}_{\text{locked}})$$

where the min is taken over all floating variables other than i and where floating variable i is locked to value $\varkappa_i$.

Since each continuous/range/discrete distance contributes additively to the total distance, we can rank the possible values for any given floating variable i according to $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i) + d_1^{(i)}(v_i)$. The best (according to this heuristic) setting for floating variable i is

$$\varkappa_i^* = \arg\min_{\varkappa_i} \left\{Z_{\mathrm{l,f}}^{(i)}(\varkappa_i) + d_1^{(i)}(v_i)\right\}.$$

The heuristic then suggests a variable ordering in order of increasing $Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*) + d_1^{(i)}(v_i^*)$ so that we first select the floating variable with the lowest sum and last select the variable with the highest sum.

Finally, we turn to the lower bound on the distance. Noting that the continuous and range distance contribution does not have the form of a sum over the dimensions (as the discrete contribution does), we write the lower bound, lb, as[22]

$$\mathrm{lb} = Z_{\mathrm{l,l}} + \sum_i \left(Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*) + \delta_{i,i^*} d_1^{(i)}(\varkappa_i^*)\right) + Z_{\mathrm{f,f}}^*$$

where $i^* = \arg\min_i \left(Z_{\mathrm{l,f}}^{(i)}(\varkappa_i^*) + d_1^{(i)}(v_i^*)\right)$ (i.e., i^* labels the floating dimension with the least $Z_{\mathrm{l,f}}^{(i)} + d_1^{(i)}$ contribution to the distance).

[22] $\delta_{x,x'}$ is the Kronecker delta function which is 1 if and only if $x = x'$.

E.2.2 Improvements

We point out that an extensive literature exists on parallel branch and bound algorithms. Many of the techniques developed are applicable to the present problem.

DO NOT SCALE